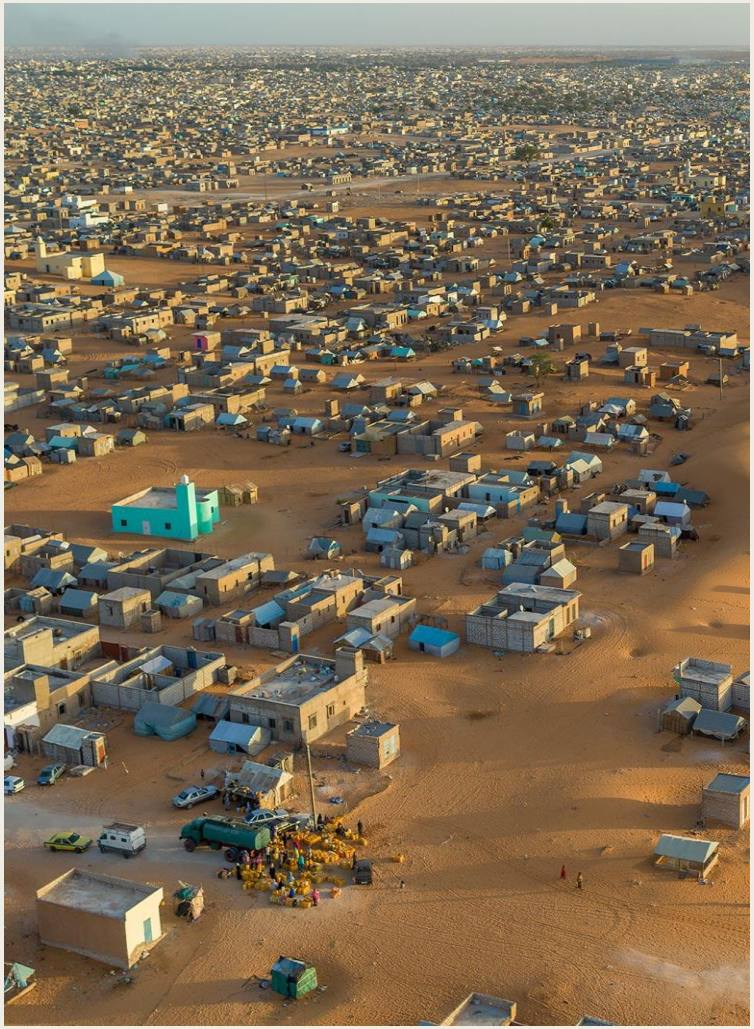
ETUDE DES BARKHANES À PETITE ÉCHELLE ET MODÉLISATION

Un premier pas vers la lutte contre l'ensablement

Motivation

- Déplacement des dunes
  - Facteur d'expansion des déserts
  - Destruction d'infrastructures

- Mieux comprendre les dunes pour lutter contre l'ensablement :
  - Simulation informatique
  - Modélisation à petite échelle



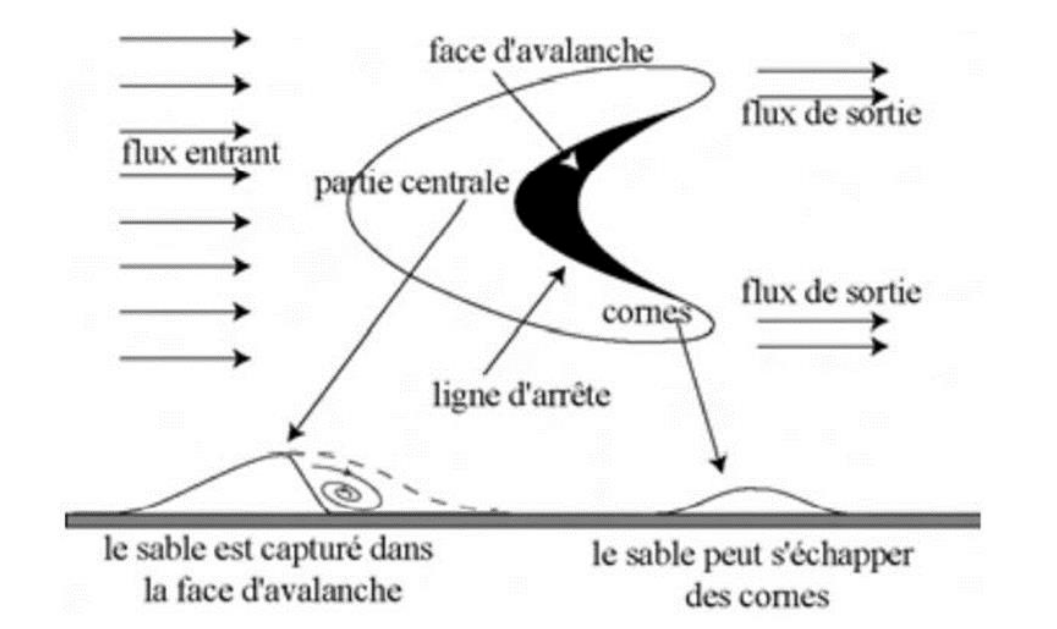
Nouakchott en Mauritanie

Image : George Steimetz, National Geographic

Sommaire

I – Morphologie et dynamique des barkhanes

A - Morphologie



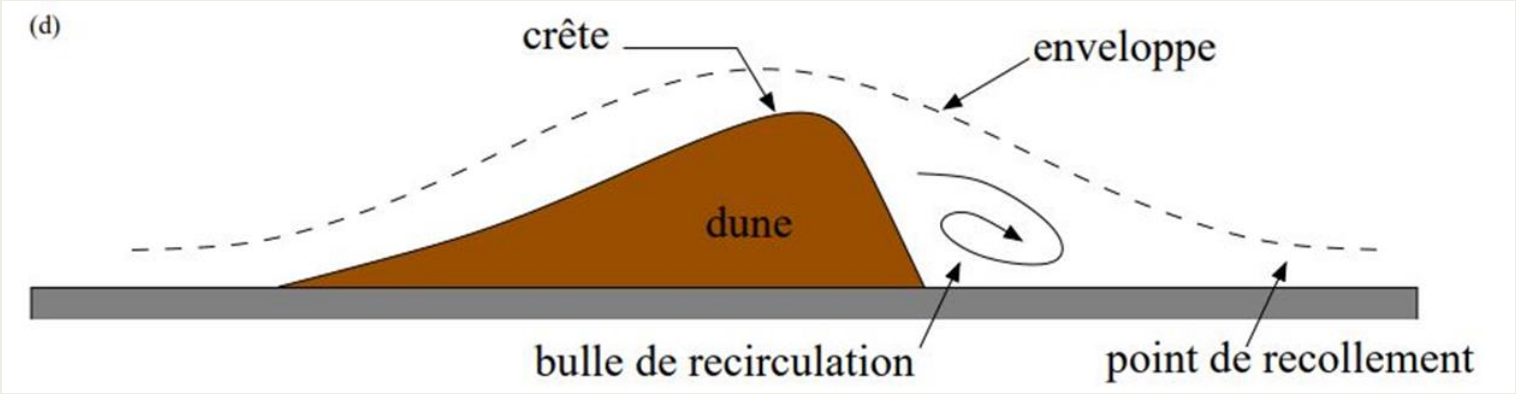
- Proportionnalité entre les dimensions
  - Longueur ~ Largeur
  - Hauteur ~ 1/10*Largeur

- Forme de croissant
- Formée dans des zones de vent unidirectionnel

Image : Pascal Hersen, thèse de doctorat

I – Morphologie et dynamique des barkhanes

B – Dynamique



Ecoulement autour d'une dune : accéleration du vent puis dépôt du sable sur la face d'avalanche

- Longueur de saturation du flux Lsat $\propto$ Ldrag = $\frac{\rho grain}{\rho fluide}$*d

- Loi de vitesse $c = \frac{q_c}{h}$ puis, pour une barkhane $c = \frac{K}{m\hat{}(\frac{1}{3})}$

Image : Amandine Garcia, thèse de doctorat

II – Modélisation à petite échelle

A – Montage expérimental

- Ldrag = $\frac{\rho grain}{\rho fluide}$*d → Réduire la taille minimale : on se place dans l'eau

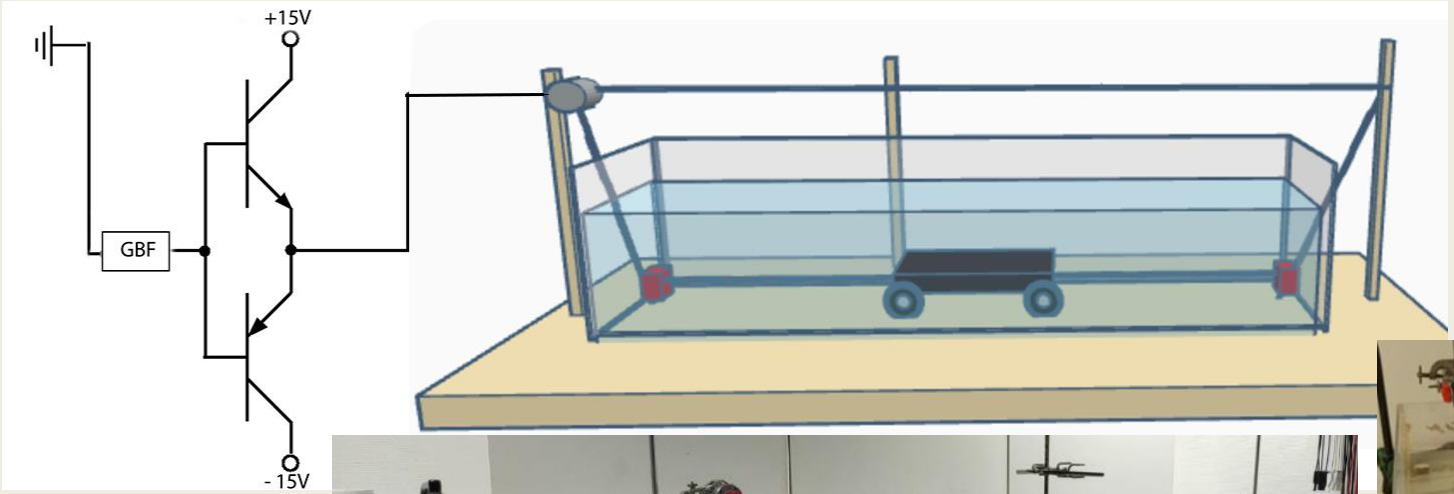
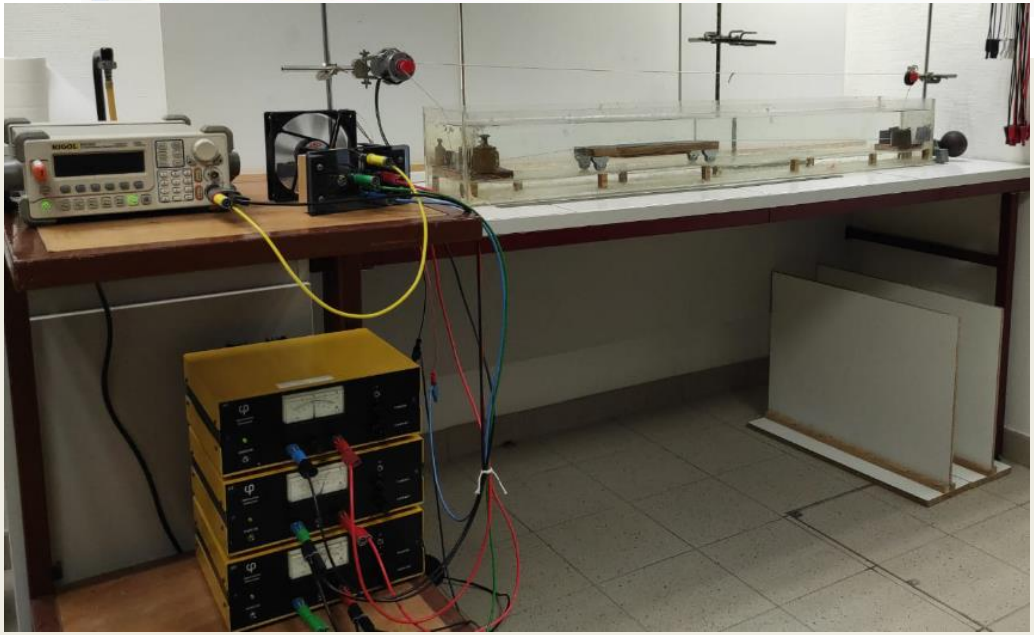
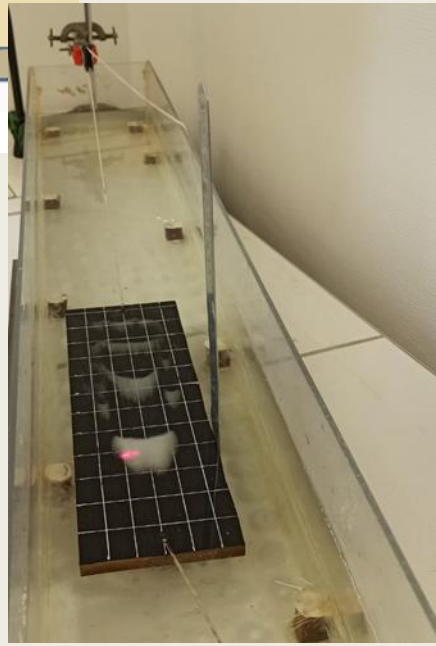


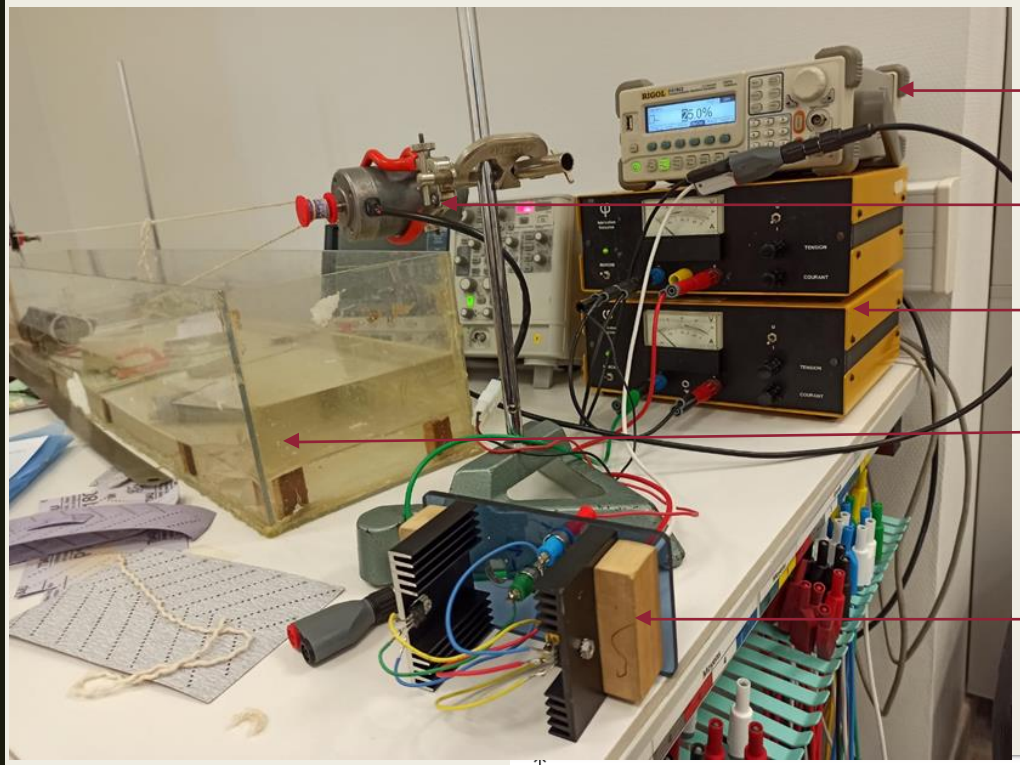
Schéma du montage



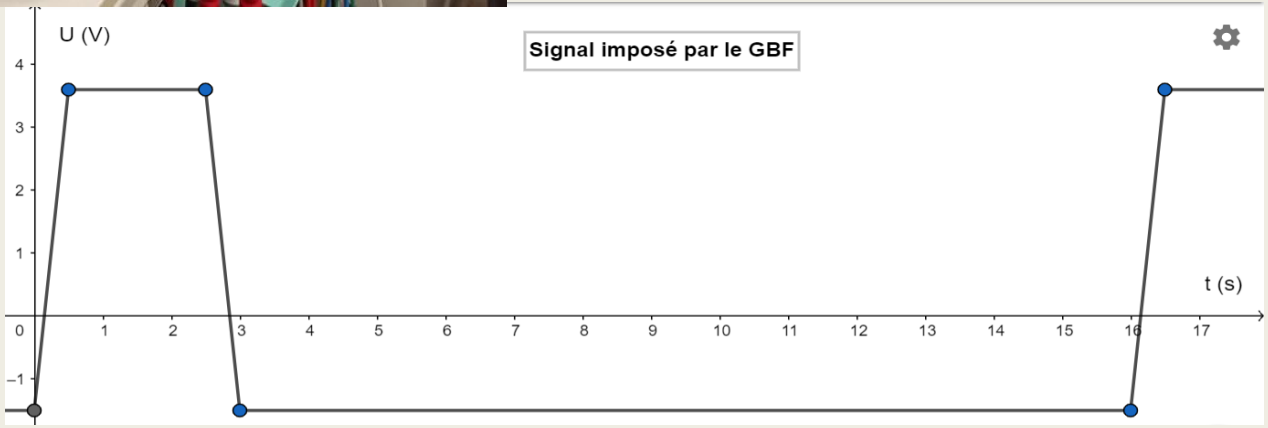


II – Modélisation à petite échelle

A – Montage expérimental

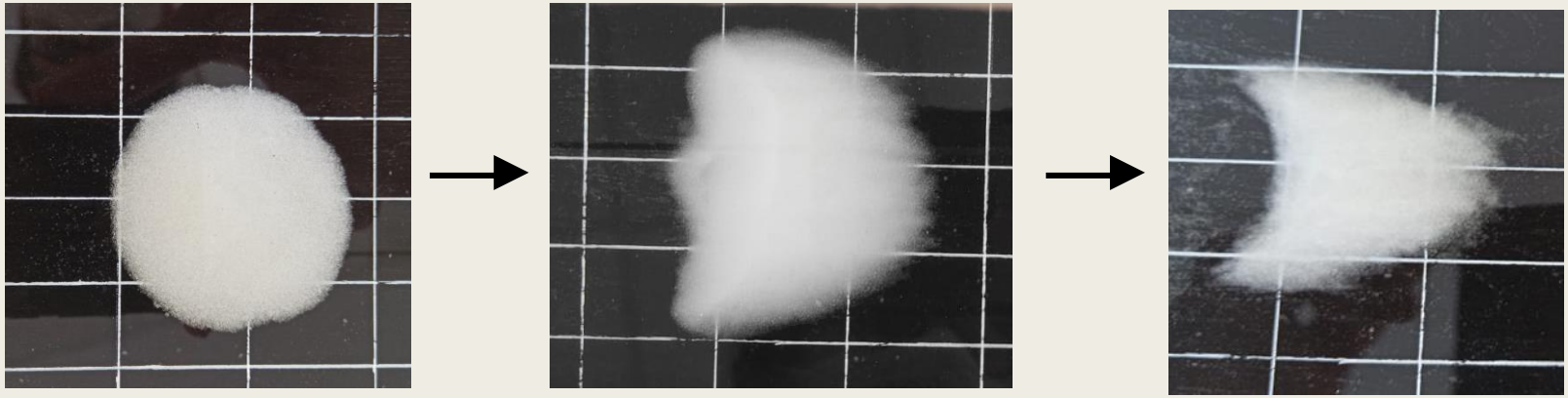


- Aller rapide, retour lent, pour simuler un courant unidirectionnel



II – Modélisation à petite échelle

B – Formation de la barkhane



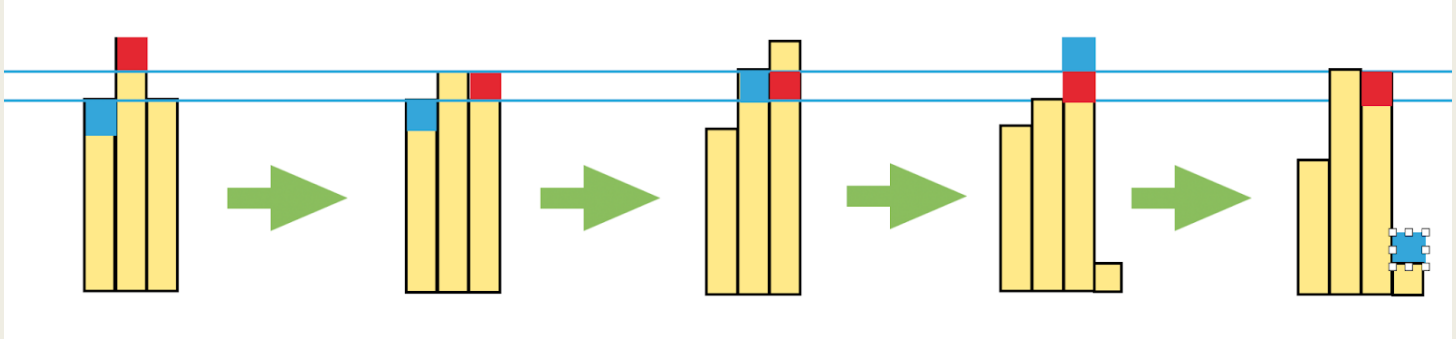
Tas conique initial

20 aller-retours

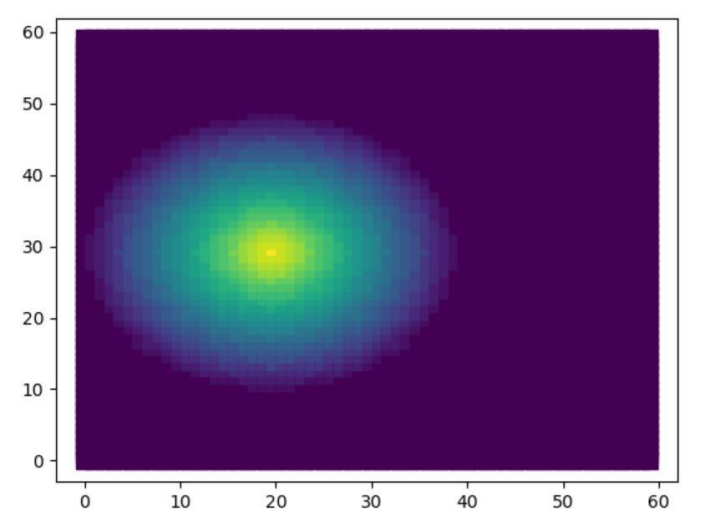
40 aller-retours

III – Simulation informatique

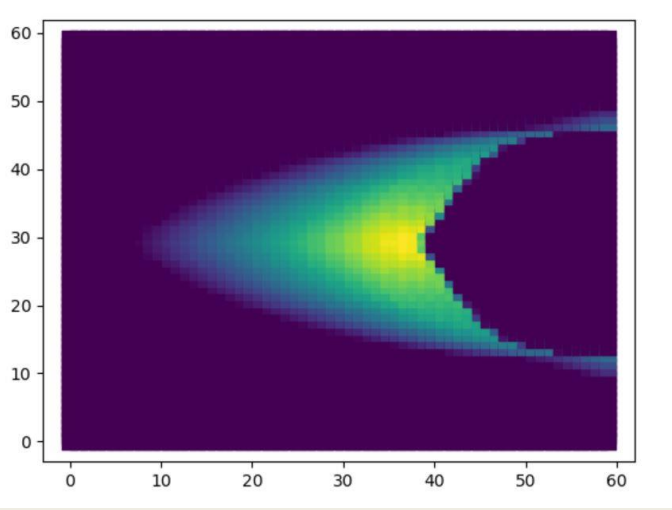
- Automate cellulaire codé en Python



Règle de déplacement des grains



800 itérations



IV – Etude des résultats

A – Morphologie & Rapports constants

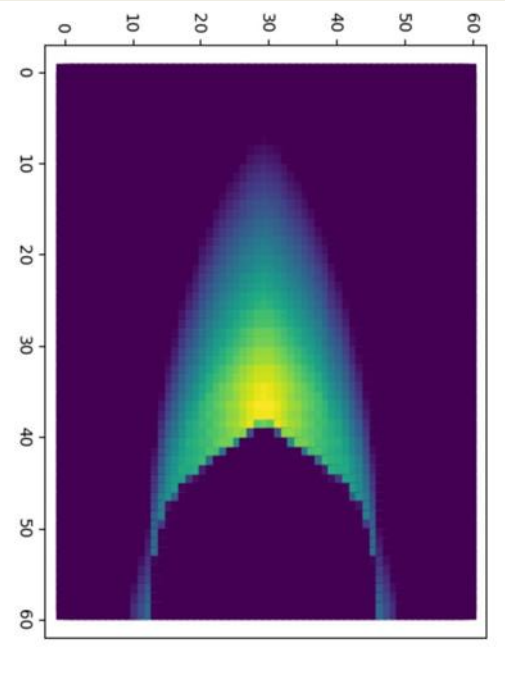
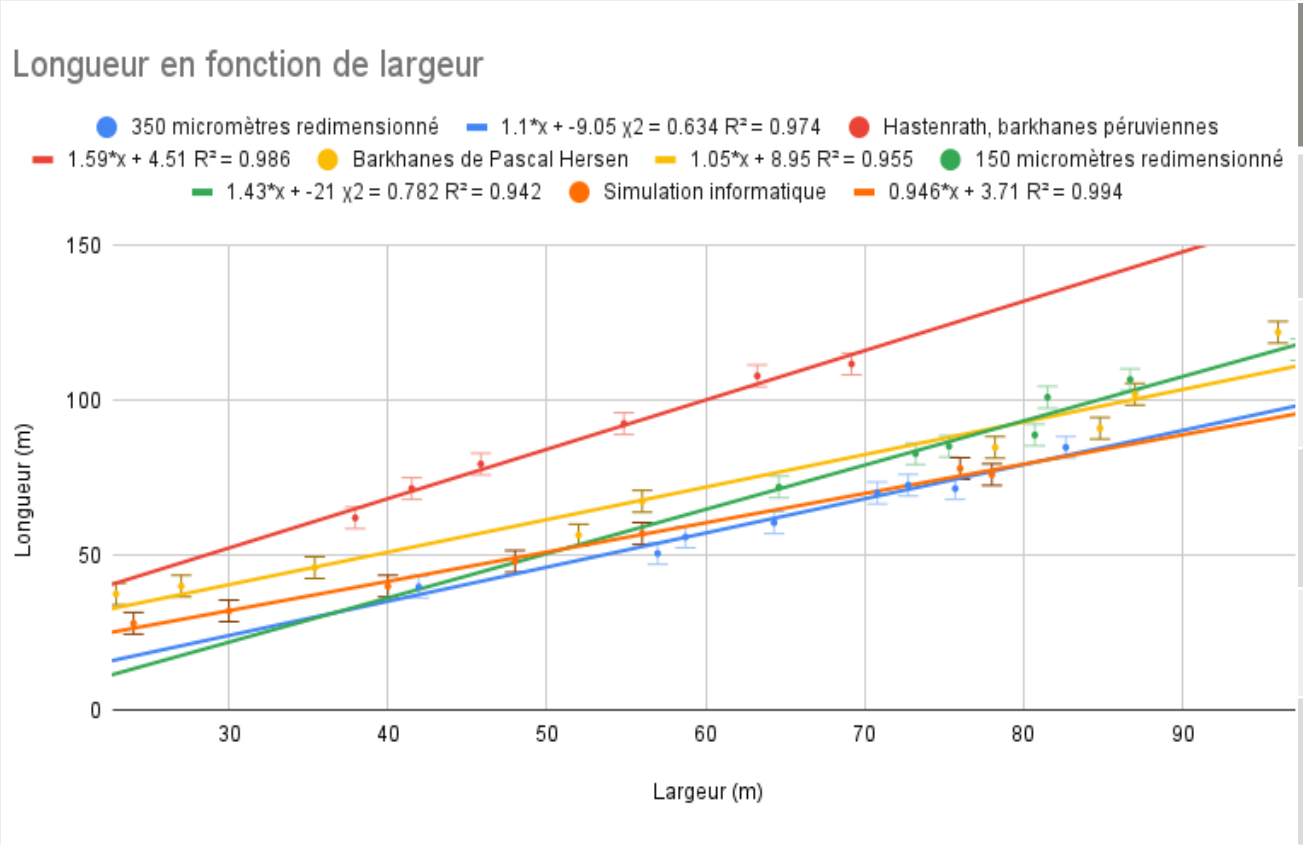


Image : Google Maps (champ de dunes près de Laâyoune en Mauritanie)

IV – Etude des résultats

A – Morphologie & Rapports constants



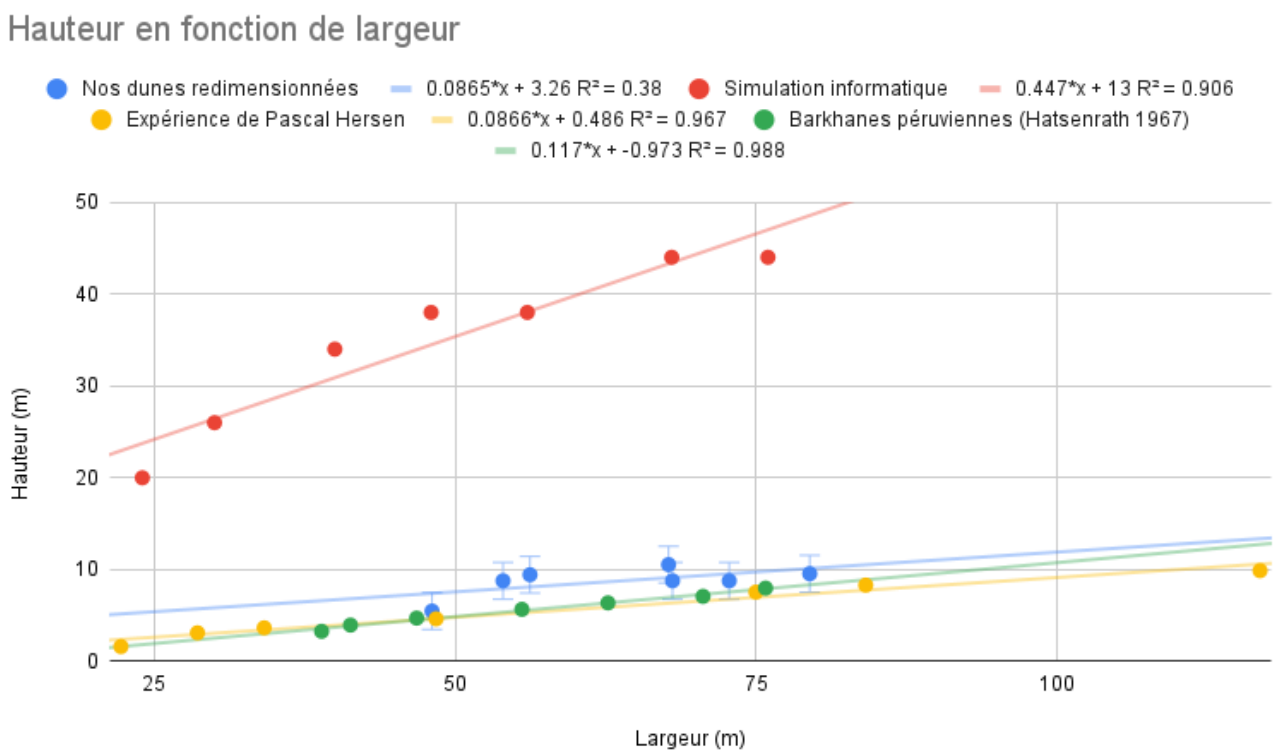
Dunes	Rapport longueur/ largeur
Nos dunes redimensionnées 350µm	1.1
Nos dunes redimensionnées 150µm	1.43
Expérience de Pascal Hersen	1.05
Simulation Informatique	0.95
Dunes Réelles (Hastenrath 1967)	1.59

- Les barkhanes centimétriques vérifient les lois d'échelle quel que soit leur diamètre : modèle validé
- La simulation numérique produit des résultats cohérents

IV – Etude des résultats

A – Morphologie & Rapports constants

- Difficultés rencontrées :
  - Manque de précision avec le laser
  - Rapport d'aspect trop important pour les dunes simulées

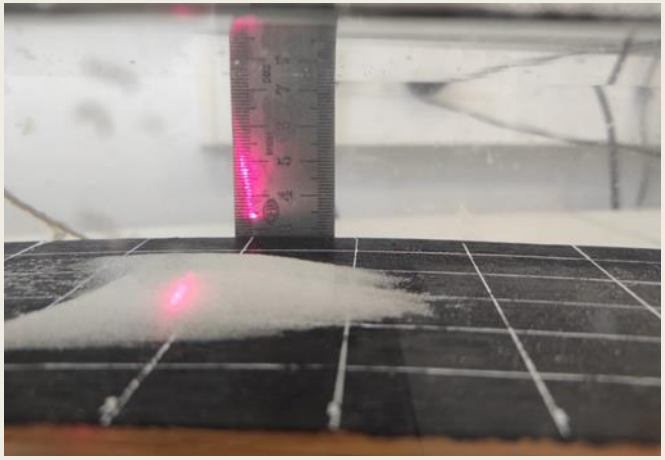


Dunes	Rapport H/L
Nos dunes redimensionnées	0.09
Expérience de Pascal Hersen	0.0866
Simulation Informatique	0.447
Dunes Réelles (Hastenrath 1967)	0.117

IV – Etude des résultats

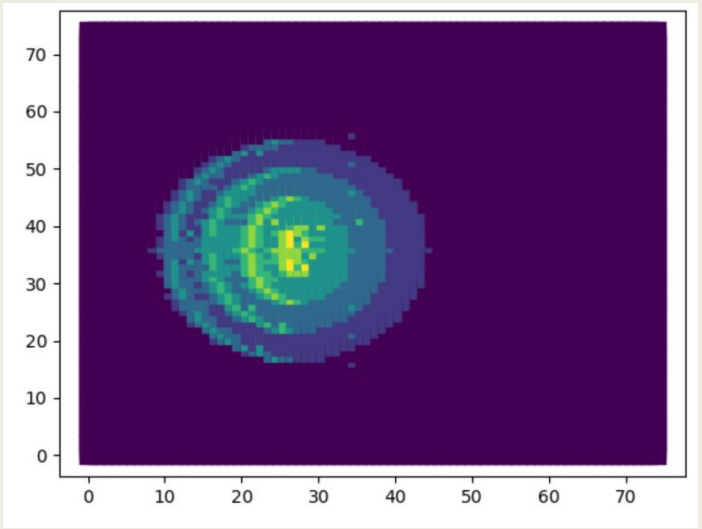
A – Morphologie & Rapports constants

- Difficultés rencontrées



Mesure avec laser
manque de précision

Dune formée à partir d'un tas conique (50,5)

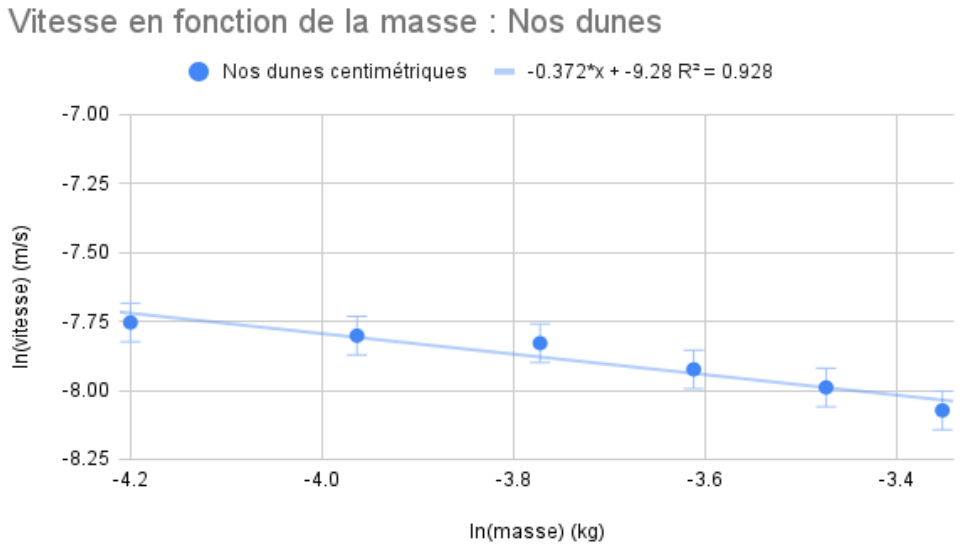


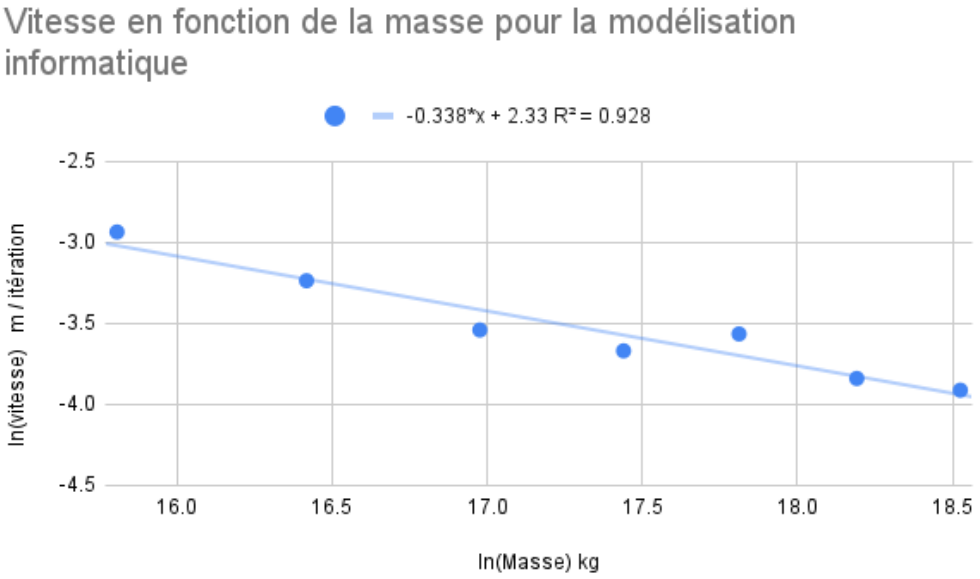
Limites dû au caractère discret de la modélisation :
→ Tendance à accentuer la hauteur
→ Dunes mal formées pour des rapports d'aspect faibles

Solution : dunes plus grandes → plus proche de la réalité

IV – Etude des résultats

B – Déplacement en fonction de la masse





- On retrouve la loi de vitesse théorique $c = \frac{K}{m\hat{}(\frac{1}{3})}$

Dunes	Vitesse moyenne
Dunes réelles	1-70 m/an
Nos dunes centimétriques	28m/jour
Expérience Hersen	7cm/jour
Modélisation informatique	2m/itération

Vitesses mesurées très différentes:

- Déplacement de la plaque trop rapide
- Transport sédimentaire plus faible à petite échelle
- Impossible de faire le lien entre itérations et temps

Conclusion

- Simulation informatique et modélisation à petite échelle sont des modélisations cohérentes
- Redimensionnement par Idrag → Lien avec les dunes réelles.
- Mieux comprendre les dunes pour prévenir contre l'ensablement
- Possibilités futures : étude des champs de dunes, collisions ?

Annexe : programme Python

```python
import numpy as np
import matplotlib.pyplot as plt
import random as rd

def tas_conique(diametre , hauteur):

    taille_matrice = diametre*3//2
    rayon=diametre//2
    pente = hauteur/rayon
    milieu = taille_matrice//2

    mat = np.zeros((taille_matrice,taille_matrice))

    for ind_ligne in range(rayon): #les n premieres lignes avec n = diametre
        for ind_col in range(rayon):
            distance = np.sqrt(ind_ligne**2+ind_col**2) #distance par rapport au milieu

            if distance <= rayon:
                empilement = int(hauteur - distance*pente)
                mat[milieu - ind_ligne][milieu - ind_col] = empilement
                mat[milieu + ind_ligne][milieu - ind_col] = empilement
                mat[milieu - ind_ligne][milieu + ind_col] = empilement
                mat[milieu + ind_ligne][milieu + ind_col] = empilement
    return mat


#REGLES
#La regle unidirectionnelle concerne le déplacement des grains du au courant avant le sommet
#La regle effondrement concernent davatange l'effondrement qui intervient après le sommet, elle est complémentaire des regles 'courant'

def regle_unidirectionnelle(mat):

    new_mat = mat.copy()
    for ligne in range(1,len(mat)-1):
        for colonne in range(len(mat)-1):

            if (mat[ligne-1][colonne] < mat[ligne][colonne]) and (mat[ligne+1][colonne] <= mat[ligne][colonne] + 1) :
                #Contraintes sur les hauteurs de tas de devant et derrière
                new_mat[ligne][colonne] = new_mat[ligne][colonne] - 1
                new_mat[ligne+1][colonne] += 1

    return new_mat
```

```python
def nombre_de_grains_qui_tombent(delta_hauteur):
    nb_effondres = int((delta_hauteur+2.0)/2 * rd.random()) + 1 #D'apres le sujet 0 en info à Mines - Pont
    return nb_effondres

def regle_effondrement(mat):
    mat_new = mat.copy()
    for i in range(len(mat)-1):
        for j in range(len(mat)):
            delta_hauteur = mat[i][j]-mat[i+1][j]

            if delta_hauteur > 1:
                grains_qui_tombent = nombre_de_grains_qui_tombent(delta_hauteur)
                mat_new[i][j] -= grains_qui_tombent
                mat_new[i+1][j] += grains_qui_tombent

    return mat_new

#Programme pour appliquer les règles

def application_regle(mat, ite):
    mat_finale = mat.copy()
    for k in range(ite):
        mat_finale = regle_effondrement(regle_unidirectionnelle(mat_finale))

    return mat_finale


#Anayse et determination de caracteristiques

def hauteur_barcane(mat):
    return max([max(liste) for liste in mat])

def coord_hauteur(mat):
    ind_ligne = 0
    maxi = hauteur_barcane(mat)

    for i in range(len(mat)):
        if max(mat[i]) == maxi:
            ind_ligne = i
    return ind_ligne
```

```
083| def longueur(mat):
084|     taille_matrice = len(mat)
085|     ind1,ind2 = 0,taille_matrice - 1
086|     check1 = 0
087|     check2 = 0
088|     for ind_ligne in range(taille_matrice-1):
089|
090|         ligne1,ligne2 = mat[ind_ligne],mat[ind_ligne+1]
091|
092|         if max(ligne1)==0 and max(ligne2)> 0 and check1 == 0 :
093|             ind1 = ind_ligne +1
094|             check1 = 1
095|         if max(ligne1)>0 and max(ligne2) ==0 and (ind1 != 0) and check2 == 0:
#pour avoir un ind2 plus grand que ind1
096|             ind2 = ind_ligne
097|             check2 = 1
098|     longueur = ind2 - ind1 + 1
099|     return longueur
```

```
101| def largeur(mat):
102|     mat_transpo = mat.transpose()
103|     largeur = longueur(mat_transpo)
104|     return largeur
105|
106| def largeur3(mat): #programme qui mesure la mauvaise largeur car celle-ci est
mesurée sur la ligne où la dune est de hauteur maximale, mais permet d'éviter les
problèmes des cornes eventuelles qui faussent la largeur de la dune mesurée dans le
programme largeur)
107|     cote_gauche, cote_droit = 0,0
108|     i_hauteur = coord_hauteur(mat) #numero de ligne de la hauteur
109|
110|     for j in range(len(mat)-1):
111|         if mat[i_hauteur][j] ==0 and mat[i_hauteur][j+1] !=0:
112|             cote_gauche = j
113|         elif mat[i_hauteur][j] !=0 and mat[i_hauteur][j+1] ==0:
114|             cote_droit = j
115|     return cote_droit - cote_gauche +1
116|
```

```python
def affichage_2D(mat):

    abscisse =[]
    ordonnee = []
    empilements = []
    taille_matrice = len(mat)
    for i in range(taille_matrice):
        for j in range(taille_matrice):
            abscisse.append(i)
            ordonnee.append(j)
            empilements.append(mat[i][j])


    plt.scatter(abscisse, ordonnee,c=np.array(empilements),s=100,marker = 's')
    plt.show()

#Expérience

def experience_unique(diametre, hauteur, ite):

    #Execution
    A = tas_conique(diametre , hauteur)
    taille_matrice = (diametre*3)//2
    rayon = diametre//2
    masse = 2500*np.pi*(hauteur/3)*(rayon)**2 #à partir du volume d'un cone

    affichage_2D(A)
    print("Etat initial")
    print("Taille matrice : ", taille_matrice)
    print("Diametre : ", diametre)
    print("Hauteur : ", hauteur)
    print("Masse : ", masse)
    coordhauteur1 = coord_hauteur(A)

    B = application_regle(A,ite)
    affichage_2D(B)
    print("Etat final")
    print("Itérations : ", ite)
    print("Hauteur : ",hauteur_barcane(B))
    print("Longueur : ", longueur(B))
    print("Largeur : ", largeur3(B))
    coordhauteur2 =  coord_hauteur(B)
    distance = coordhauteur2 - coordhauteur1
    print("Distance parcourue : ", distance)
```

```
def experience_serie_distance(diametre,hauteur):

    A = tas_conique(diametre , hauteur)
    taille_matrice = 75 # arbitraire, choisi de telle sorte que les dunes ne sortent pas du cadre de la matrice au bout d'un certain temps et faussent les valeurs
    rayon = diametre//2
    masse = 2500**np.pi*(hauteur/3)*(rayon)**2
    coordhauteur1 = coord_hauteur(application_regle(A, 100)) #formation de la dune initiale

    distances = []

    for i in range(1, 18): #arbitraire, 18 valeurs obtenues
        ite = 50*i #arbitraire, on ajoute 50 pour chaque dune
        B = application_regle(A,ite)
        coordhauteur2 = coord_hauteur(B)
        distance = coordhauteur2 - coordhauteur1 #mesure distance par la difference de coordonnée des points de la hauteur
        distances.append(distance)

    return distances

def experience_serie(depart_diametre,fin_diametre,pas):

    #Parametrage interne
    taille_matrice = 2*fin_diametre
    ratio_hd = 1/3 # peut être modifié mais un rapport d'aspect trop faible (plus proche de la réalité, 1/10 par ex.) ne permet pas de former des dunes correctes
    #Execution
    longueurs, largeurs, hauteurs = [],[],[]
    numero_du_tas = 0

    for diametre in range(depart_diametre,fin_diametre,pas):
        numero_du_tas += 1

        intervalle = (fin_diametre - depart_diametre)/pas

        A = tas_conique(diametre , int(diametre*ratio_hd))
        A = application_regle(A, int(diametre*5)) # nombre d'itérations arbitraire, ici choisi adapté pour que la dune ne sorte pas de l'affichage (on peut modifier taille_matrice sinon aussi) et pour qu'elle garde une forme cohérente (les dunes ne sont pas stables, elles se désagrègent si trop d'itérations)
        affichage_2D(A)
        print(numero_du_tas,"/", intervalle)
        longueurs.append(longueur(A))
        largeurs.append(largeur(A))
        hauteurs.append(hauteur_barcane(A))


    plt.plot(longueurs,largeurs)
    plt.xlabel("Longueur")
    plt.ylabel("Largeur")
    plt.show()

    plt.plot(longueurs,hauteurs)
    plt.xlabel("Longueur")
    plt.ylabel("Hauteur")
    plt.show()

    return hauteurs, longueurs, largeurs

```